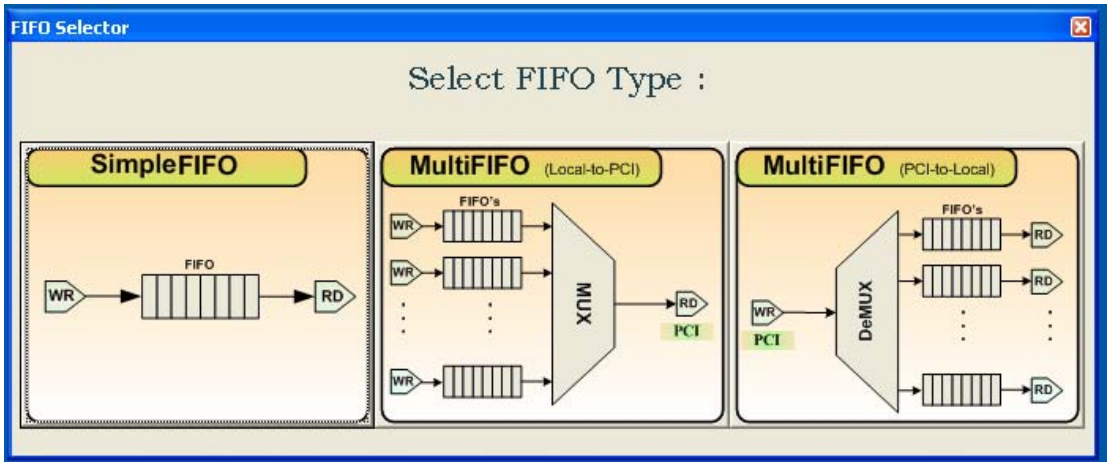
PROCMegaFIFO IP



User's Guide

September 2008

GiDEL products and their generated products are not designed, intended, authorized, or warranted to be suitable for use in life-support applications, devices or systems or other critical applications

© 1993 - 2008 by ***GiDEL*** Ltd. All rights reserved. ***GiDEL***, ***PROCStar II***™, ***PROCSpark II***™, ***PSDB***™, ***PROCWizard***™, ***PROCCamLink***™, ***PROCMultiPort***™, ***PROCMegaFIFO***™ and other product names are trademarks of ***GiDEL*** Ltd., which may be registered in some jurisdictions. This information is believed to be accurate and reliable, but ***GiDEL*** Ltd. assumes no responsibility for any errors that may appear in this document. ***GiDEL*** reserves the right to make changes in the product specifications without prior notice.
Windows NT, Windows XP, Windows 2000, Stratix II, EP2S60, DDRII, CameraLink and other brands and product names are trademarks or registered trademarks of their respective holders.

USA
1600 Wyatt Drive Suite 1
Santa Clara,
CA 95054, USA
Tel: 1 - 408 - 969 - 0389
Fax: 1 - 866 - 615 - 6810

sales_usa@GiDEL.com

Worldwide
2 Ha'ilan Street, P.O. Box 281
Or Akiva,
Israel 30600
Tel: +972 - 4610 - 2500
Fax: +972 - 4610 - 2501

sales_eu@GiDEL.com

Web: www.GiDEL.com
info@GiDEL.com

Contents

Introduction 1

PROCMegaFIFO **Key Features** 2

Setting Up ***PROCMegaFIFO*** **Using** ***PROCWizard*** 3

Using ***PROCMegaFIFO*** 11

General Notes 11

Working with SimpleFIFO 11

Resetting and starting SimpleFIFO: 12

Reset from Hardware: 12

Reset from Software: 12

Accessing SimpleFIFO from hardware: 12

Writing to FIFO: 12

Reading from FIFO: 12

Accessing SimpleFIFO from software: 13

Working with limited packets (not an infinite stream): 13

LIMITATIONS 14

Working with MultiFIFO 14

Resetting and starting MultiFIFO: 14

Accessing MultiFIFO from hardware: 14

Writing to MultiFIFO with multiple write ports (Local-to-PCI) : 14

Reading from MultiFIFO with multiple read ports (PCI-to-Local) : 14

Accessing MultiFIFO from software: 15

PROCMegaFIFO **Parameters and Signals** 18

Parameters for SimpleFIFO 18

Signal/Bus Names for SimpleFIFO 19

Parameters for MultiFIFO 20

Signal/Bus Names for MultiFIFO 20

Figures

Figure 1 – Configuration Mode Icon 3
Figure 2 – Accessing MegaFIFO in IC Component Drop-Down Menu 4
Figure 4 – The FIFO Selector 5
Figure 5 – SimpleFIFO Properties Dialog 6
Figure 6 –Simple FIFO Configuration Screen 7
Figure 7 – MultiFIFO Properties Dialog 8
Figure 8 – MegaFIFO Ports List 9
Figure 9 – MegaFIFO Port Settings 9
Figure 10 – Set MegaFIFO Ports List 10

Tables

Table 1 – CMultiFIFO Simple Mode Methods 15
Table 2 – CMultiFIFO Effective Mode Methods 16
Table 3 – CMultiFIFO General Methods 17
Table 4 - SimpleFIFO Parameter Names 18
Table 5 – SimpleFIFO Signal and Bus Names 19
Table 6 – MultiFIFO Parameter Names 20
Table 7 – MultiFIFO Signal and Bus Names 21

Introduction

PROCMegaFIFO*™** IP is a GiDEL Intellectual Property that provides a simple and convenient way to transfer data to / from GiDEL ***PROC boards. With ***MegaFIFO***, data may be transferred between the host PC and user's subdesigns, or between subdesigns, using the on-board memory, or SODIMM memory module, as a very large FIFO.

MegaFIFO eliminates the need to take care of synchronization when transferring data between designs running at different clocks. The software no longer needs to respond to the hardware in real-time and hardware designs may now transfer data in bursts and withdraw it in a continuous stream.

MegaFIFO uses special arbitration techniques when transferring data between the host PC and user's subdesigns. These techniques prevent memory overflows / underflow, thus using the maximum available bandwidth for data transfers.

MegaFIFO is simple to use. Its straightforward interface is based on the familiar PROCMultiPort interface. Request and Acknowledge signals ensure correct data transfers. On the software side, the Proc class methods perform automatic initialization of the FIFO logic and enable easy data transfers using DMA.

PROCMegaFIFO Key Features

***GiDEL PROCMegaFIFO*™** IP was designed to provide an effective and simple means of using the on-board memory space. MegaFIFO defines the on-board memory (or a part of it) as a FIFO storage or as a synchronization buffer. The main advantage of MegaFIFO is its ability to work with huge memory banks and define large FIFO memories in various configurations. The FIFO memory bank may be on-board DDR memory or memory modules connected to SODIMM sockets.

MegaFIFO uses cyclic memory pointers. This way, a virtual infinite memory space is created, assuming that the memory bandwidth is not exceeded.

MegaFIFO IP consists of two IPs:

- **SimpleFIFO** has two data ports: a source **write** port and a destination **read** port. Simple FIFO behaves as a large FIFO buffer between those ports. Each of the ports can be connected to user's subdesign or to PCI (host software).
- **MultiFIFO** makes possible to send or receive large amounts of data from or to the host software. MultiFIFO has a single port connected to the host PC and multiple ports connected to user's subdesigns. The data is transferred from / to the subdesigns using a single DMA channel to eliminate situations when several DMA channels are competing on the PCI channel. This way, the highest data transfer rate is achieved.

MegaFIFO IP key features include:

- ✓ User-friendly design with standard input and output interface
- ✓ Optimized for easy control from hardware and software
- ✓ Large memory buffers managed using cyclic memory pointers
- ✓ High data transfer rates using a single DMA channel for multiple transfers to ensure maximum performance
- ✓ Automatic data transfer rate management to avoid buffer underflow and overflow

Setting Up *PROCMegaFIFO* Using *PROCWizard*

Important

1. At least one of MegaFIFO ports is always connected to a user's subdesign. You should have your source / destination subdesigns ready and defined **before** you start setting up the MegaFIFO.
2. In the current version of PROCDeveloper's Kit, it is not be possible to put a MegaFIFO in the same memory bank as a MultiPort, and vice versa.

You should have your target subdesign ready before you start setting up the MegaFIFO.

The following should be defined:

- IC(s)
- Main clock and memory clock
- User designs and with appropriate clocks

To learn how to define subdesigns, please refer to ***PROCWizard*** User's Manual.

To set up PROCMegaFIFO using PROCWizard:

1. Click the button in PROCWizard's toolbar, shown in the figure below, to enter the Configuration Mode in GiDEL PROCWizard.

Figure 1 – Configuration Mode Icon

The Configuration Mode is used to build designs in PROCWizard. New items can be added to the design in this mode and existing items can be redesigned.

To learn more about configuration mode, please refer to ***PROCWizard*** User's Manual.

2. Click the "**+**" sign to the left of the **Designs Card [*proc card type*]** item to expand it.

3. Right click on the name of the desired IC, to access the IC component drop-down menu and select **GiDEL IP Core → MegaFIFO**.

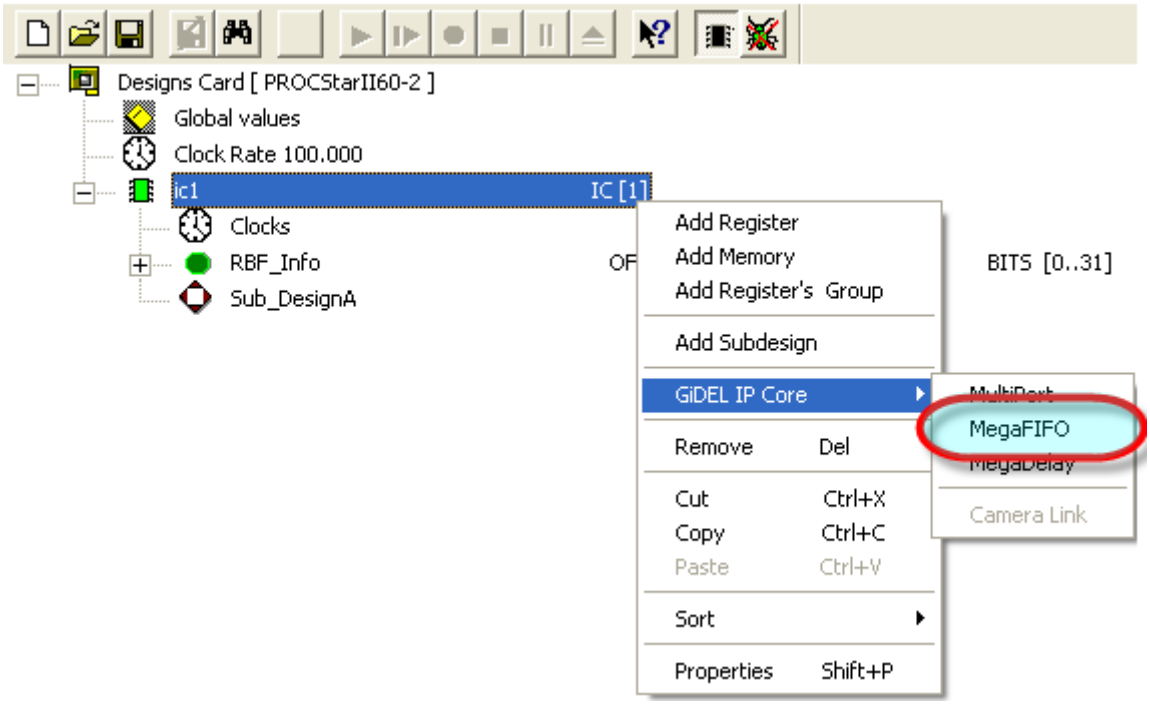


Figure 2 – Accessing MegaFIFO in IC Component Drop-Down Menu

The **FIFO Selector** appears:

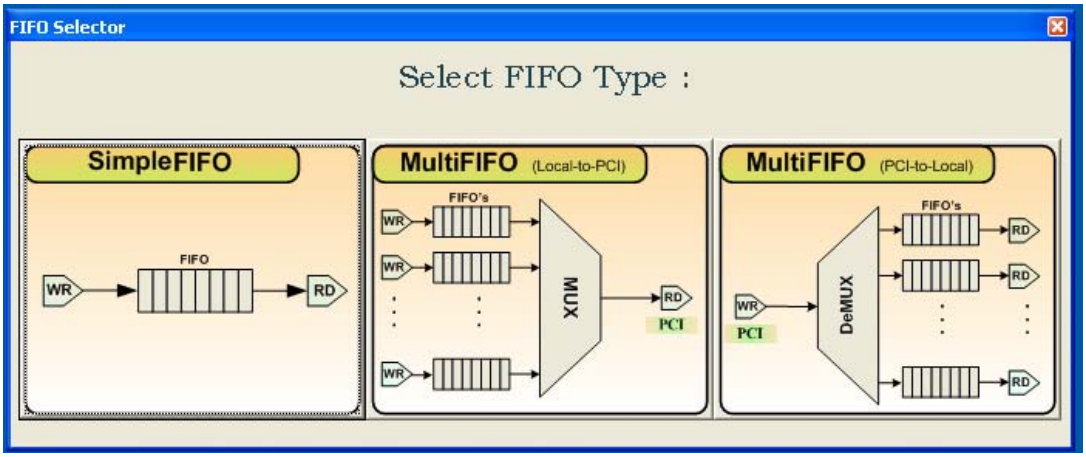


Figure 3 – The FIFO Selector

The **FIFO Selector** provides two FIFO types:

- **SimpleFIFO**
- **MultiFIFO** (can be Local-to-PCI or PCI-to-Local)

A **SimpleFIFO** may be used to transfer data between a PCI PORT and user's subdesign or between two subdesigns. This FIFO type always has one write port and one read port and has a very simple interface.

A **MultiFIFO** is used to transfer data between the software and one or several of user's subdesigns. When users wish to transfer data between the software and several subdesigns, it is always better to use a MultiFIFO and not several SimpleFIFO. This will significantly enhance performance and reduce usage of FPGA cells.

To set up SimpleFIFO:

1. Choose **SimpleFIFO** from the **FIFO Selector**. The **SimpleFIFO** properties dialog appears.

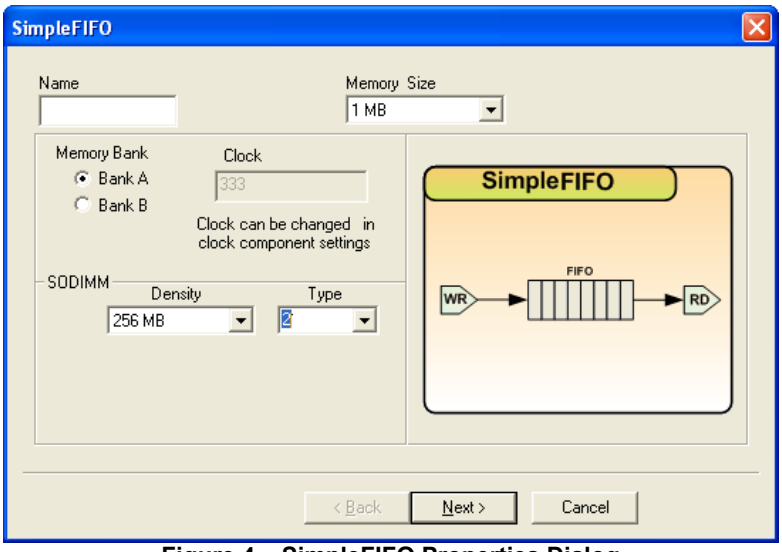


Figure 4 – SimpleFIFO Properties Dialog

2. In **SimpleFIFO** define the following properties:
   - The **Name** of the FIFO memory
   - The **Memory Size** to be allocated for the FIFO
   - The **Memory Bank** in which the FIFO will reside.

     If the selected **Memory Bank** is a SODIMM memory, then a **SODIMM** group box with the fields **Density** and **Type** will appear in the SimpleFIFO Properties dialog box(See Figure 4).
     - Click the **Density** arrow to select one of the possible memory densities.
     - Click the SODIMM **Type** arrow to select your SODIMM's memory type. To determine your SODIMM's type, refer to the ***GiDEL SODIMM Type Datasheet*** document. In accordance to the SODIMM's **Type,** the PROCWizard will automatically generate a customized design.

3. Click Next to continue the configuration of SimpleFIFO.

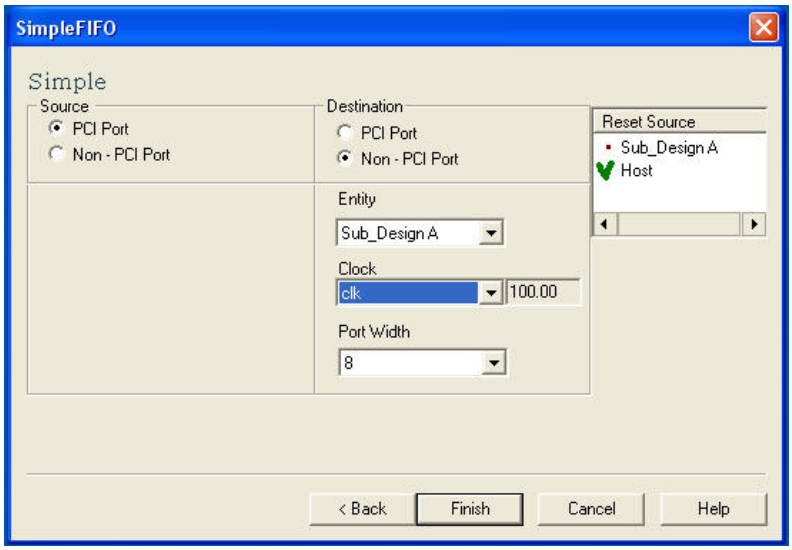


Figure 5 –Simple FIFO Configuration Screen

4. Define **SimpleFIFO** by selecting the:

   - Desired **Source** and **Destination** Ports. A PCI Port will be connected to the software and a Non-PCI port will be connected to user's subdesign.
   - A previously created subdesign **Entity** for a Non-PCI Port
   - For a Non-PCI port, define: a **Clock** to synchronize the data with**,** and the **Port Width** (the data width in bits).
   - The **Reset Source**. The Reset Source offers two options:
     - **Host** (resetting the FIFO occurs from the software)
     - **Subdesign** (resetting the FIFO occurs from the selected subdesign)

Note If a subdesign was selected as a reset source, the user has to ensure that the reset logic exists in that subdesign.

5. Click **Finish** to accept the **Simple FIFO** settings.

To set up MultiFIFO:

Choose **MultiFIFO** from the **FIFO Selector**. The **MultiFIFO** Properties dialog appears.

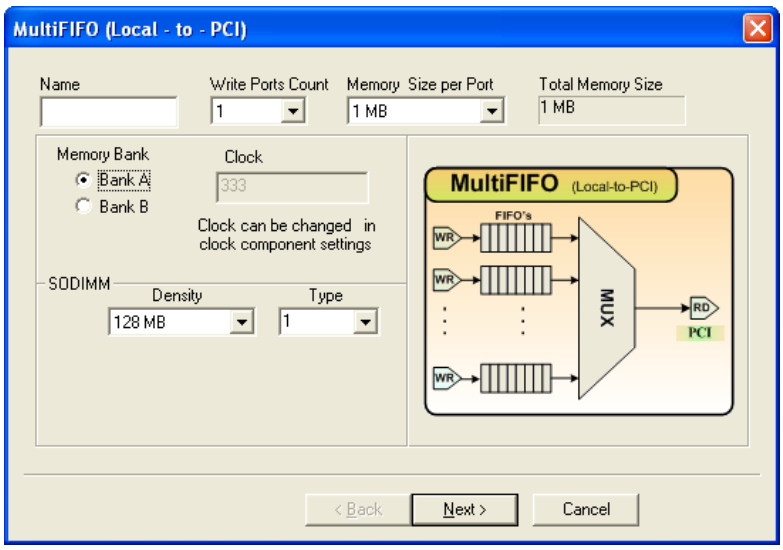


Figure 6 – MultiFIFO Properties Dialog

6. In **MultiFIFO** mode define the following:

   - The **Name** of the FIFO memory
   - The **Non-PCI** Ports Count (**Write** ports for Local-to-PCI **MultiFIFO**, or **Read** ports for PCI-to-Local **MultiFIFO**)
   - The **Memory Size per Port** (amount of memory that should be allocated for each port)
   - The **Memory Bank** in which the FIFO will reside

     If the selected **Memory Bank** is a SODIMM memory, then a **SODIMM** group box with the fields **Density** and **Type** will appear in the MultiFIFO Properties dialog box (See Figure 6).

     - Click the **Density** arrow to select one of the possible memory densities.
     - Click the SODIMM **Type** arrow to select your SODIMM's memory type. To determine your SODIMM's type, refer to the ***GiDEL SODIMM Type Datasheet*** document. In accordance to the SODIMM's **Type,** the PROCWizard will automatically generate a customized design.

Note There is a maximum of 16 ports per memory bank - this includes all MegaIP ports.

The **Total Memory Size** that appears on the right is the product of the **Non-PCI Ports Count** and the **Memory Size per Port**. This is the total amount of memory that will be allocated for the current **MultiFIFO**.

7. Click **Next** to continue the configuration of the **MultiFIFO** in the **MegaFIFO Ports List**.

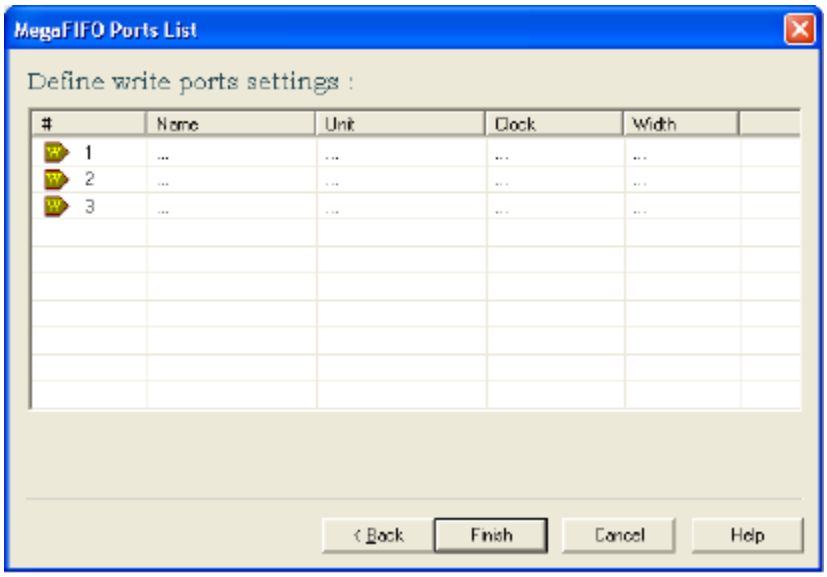


Figure 7 – MegaFIFO Ports List

8. In the **MegaFIFO Ports List**, double click on each row to access the **Port Settings** dialog window.

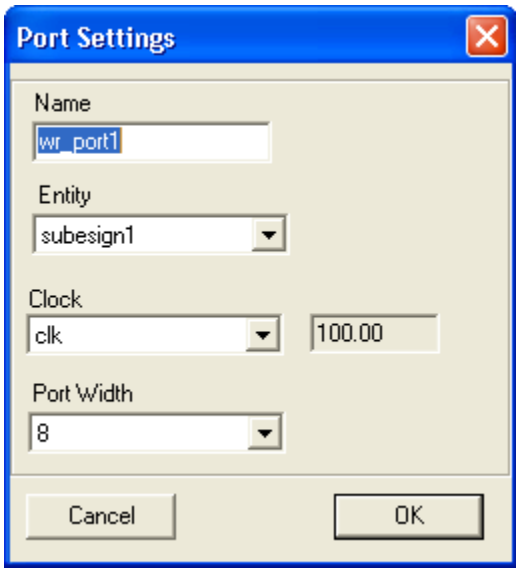


Figure 8 – MegaFIFO Port Settings

9. In Port Settings define:
    - The **Name** of the Port
    - The subdesign **Entity** to which the port should be connected
    - The **Clock** to synchronize the data with
    - The **Port Width** (data width in bits)

10. Click **OK** return to the **MegaFIFO Ports List**. Continue setting up all the ports as in the figure below.

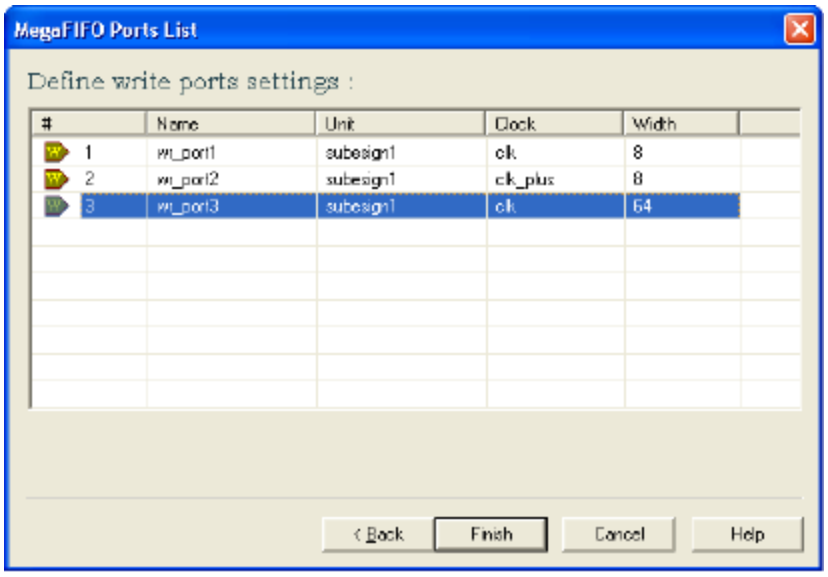


Figure 9 – Set MegaFIFO Ports List

11. Click **Finish** to accept the **MultiFIFO** settings.

General Notes

SimpleFIFO automatically adjusts the data transfer rate to avoid memory throughput overflow. Therefore, users should be aware of the possibility that data transfer may be reduced due to memory bandwidth overflow.

MultiFIFO controls the transfer rate from the PCI side only. User's logic that works with non-PCI ports of **MultiFIFO** should take care of the memory bandwidth.

To calculate the memory throughput, multiply memory clock rate by memory width and reduce it to 75%.

For example: 32bit-bus memory runs on 150MHz:

Throughput ≈ 0.75*(32*150*10^6) = 3.6Gbps

If the memory is of DDR type, the final throughput is multiplied by 2:

Throughput ≈ 3.6Gbps * 2 = 7.2Gbps

The sum of all active ports data rates should be less than the memory throughput.

Working with SimpleFIFO

SimpleFIFO has two ports: a write (source) port and a read (destination) port. Each port may be a PCI port (connected to the host software) or a non-PCI port (connected to user's subdesign). The following connection combinations are available:

- PCI **source** port and non-PCI **destination** port : data is transferred from the host software to user's subdesign
- Non-PCI **source** port and PCI **destination** port : data is transferred from user's subdesign to the host software
- Non-PCI **source** port and non-PCI **destination** port : data is transferred from one user's subdesign to another

Note that **both** source and destination ports cannot be PCI ports.

Resetting and starting SimpleFIFO:

Reset from Hardware:

If user's subdesign was chosen as reset source, a special logic must be added to that subdesign to perform resets. To reset the FIFO the user's logic has to assert a corresponding **reset_fifo_name** signal high for at least 3 clocks (the clock domain of the reset signal was chosen during reset source definition in PROCWizard)

Reset from Software:

If the Host was chosen as reset source, any write to the **reset_fifo_name** control register from the software will reset the FIFO.

Note The **clrn** signal, which resets the whole board, will reset the FIFO logic, but won't start it. To start the FIFO, it must be reset either by hardware or by the Host.

Accessing SimpleFIFO from hardware:

Writing to FIFO:

When the data is ready on the source port data bus, assert the write request signal (**wr_req_fifo_name**) high. The request signal may be set high ***on the next clock after the reset pulse falls***. The source port will acquire data on each clock as long as both write request and write acknowledge signals are high for that port. The fall of the write acknowledge signal of the source port (**wr_ack_fifo_name**) indicates that the port is not ready to receive data. Possible reasons: bandwidth was exceeded or FIFO memory is full. In the latter case, the data should be read from the read port of the FIFO to free up the memory space.

Reading from FIFO:

After the data was written to the FIFO it may be read from the read port (destination port). The read acknowledge signal (**rd_ack_fifo_name**) rises when the data is ready for reading. Assert read request signal (**rd_req_fifo_name**) high to start reading. The destination port will output data on each clock as long as both read request and read acknowledge signals are high for that port. The fall of the read acknowledge signal of the destination port indicates the port has no more data to output. Possible reasons are:

- FIFO memory is empty
- Write port is slower than the read port (FIFO is "starving")
- Bandwidth has been exceeded

Accessing SimpleFIFO from software:

If a **SimpleFIFO** has a PCI port, this port may be accessed by software using DMA. The usage of the **SimpleFIFO** in that case is similar to that of the MultiPort. Data may be read / written to the PCI ports using standard DMA methods of the **Proc** class. To learn more about these methods, please refer to the **PROCMultiPort Data Book**, **PROCWizard Data Book** and **PROC API Data Book**.

Contrary to MultiPort ports, FIFO ports do not need to be individually reset by software before being accessed. Resetting the FIFO also resets its ports.

Important

SimpleFIFO automatically manages the data transfer rate via PCI. The DMA transfer will be automatically held when FIFO enters an overflow / underflow state and resumed when FIFO exits that state. However, in case the PCI port is a read port, the user should activate DMA only after the **almost_full** flag has risen, to ensure that there is enough data for the initial DMA transfer.

Working with limited packets (not an infinite stream):

The write request signal of the source port should remain high 1 clock after the last word has been written (thus writing one extra word to the FIFO). This action is related to the cases when port width is less than 64 bits wide. Writing an extra word to the FIFO is necessary to make sure the whole packet was transferred to the destination port.

When the read port is a PCI port, the writing logic has to assert the flush signal (**fifo_name_flush**) high after transfer of the last word, thus enabling the PCI logic to read the whole packet.

Important

After finishing the limited packet transfer, users must reset the FIFO before starting the next packet.

LIMITATIONS

The minimal single packet size for the **SimpleFIFO** is 32 words, 64 bits each.

Working with MultiFIFO

MultiFIFO has a single PCI port connected to the host software and one or more non-PCI ports connected to user's subdesigns. The data is transferred between the software and the subdesigns using a single DMA channel, thus eliminating competition on the PCI bus. The interface of the non-PCI **MultiFIFO** ports is similar to that of the non-PCI **SimpleFIFO** ports. From the software side, the data transfers are managed using methods of the **CMultiFifo** class (derived from the Proc class). Using these methods, data may be transferred to / from each non-PCI port individually.

Resetting and starting MultiFIFO:

MultiFIFO is always reset by the software. There is no way for user's subdesigns to reset **MultiFIFO**.

To reset **MultiFIFO** from the software, use the **Reset()** method of the **CMultiFIFO** class.

Note

The **clrn** signal, which resets the whole board, will reset the FIFO logic, but won't start it. To start the FIFO, it must be reset using Host Reset.

Accessing MultiFIFO from hardware:

Writing to MultiFIFO with multiple write ports (Local-to-PCI) :

In Local-to-PCI **MultiFIFO**, user's subdesign(s) write data to the FIFO and software reads this data. When the data is ready on the write port data bus, assert the request signal (**req_fifo_name_port_name**) high. The FIFO port will expect data on each clock as long as both request and acknowledge signals are high for that port. The fall of the acknowledge signal of the write port (**ack_fifo_name_port_name**) indicates that the port is not ready to receive data. Possible reasons: bandwidth was exceeded or FIFO memory is full. In the latter case, the data should be read by software to free up the memory space.

Reading from MultiFIFO with multiple read ports (PCI-to-Local) :

In PCI-to-Local **MultiFIFO**, software writes data to the FIFO and user's subdesign(s) read this data. After the data was written to the FIFO by software, it may be read from the corresponding read port when the acknowledge signal (**ack_fifo_name_port_name**) rises for that port. Assert the request signal (**req_fifo_name_port_name**) high to start reading. The read port will output data on each clock as long as both request and acknowledge signals are high for that port.

The fall of the acknowledge signal of the destination port indicates the port has no more data to output. Possible reasons are:

- FIFO memory is empty
- Write port is slower than the read port (FIFO is "starving")
- Bandwidth has been exceeded

Accessing MultiFIFO from software:

MultiFIFO always has one PCI port connected to user's software. This single port may access data from any of the non-PCI ports. This port is managed using methods of the **CMultiFifo** class. In addition, this class offers various **MultiFIFO** management methods, such as FIFO reset and setup.

CMultiFifo class has the following methods:

Simple Mode Methods			
Method Name	**Input**	**Output**	**Description**
MF_BUFFER_HANDLE **ReadToBuffer (** *LPVOID Buffer,* *DWORD BufferSize,* *DWORD *ReadSize,* *int Port* **)**	Pointer to data buffer , Size of data buffer , Pointer to variable that will hold the size of data that was read, Port number	MultiFIFO Buffer handle (NULL if error occurred)	Reads all the available data in the specified FIFO port into the **Buffer** (flushes the port)
MF_BUFFER_HANDLE **FillBuffer (** *LPVOID Buffer,* *DWORD BufferSize,* *int Port* **)**	Pointer to data buffer , Data length , Port number	MultiFIFO Buffer handle (NULL if error occurred)	Reads the specified amount of data from the specified FIFO port into the **Buffer**
MF_BUFFER_HANDLE **Write (** *LPVOID Buffer,* *DWORD BufferSize,* *int Port* **)**	Pointer to data buffer , Data length , Port number	MultiFIFO Buffer handle (NULL if error occurred)	Writes the data from the **Buffer** to the specified FIFO port

Table 1 – CMultiFIFO Simple Mode Methods

Effective Mode Methods			
Method Name	**Input**	**Output**	**Description**
MF_BUFFER_HANDLE **CreateBuffer (** *LPVOID Buffer,* *DWORD BufferSize* **)**	Pointer to data buffer (NULL for automatic allocation), Buffer size	MultiFIFO Buffer handle (NULL if error occurred)	Creates MultiFIFO buffer for the effective transfer mode
void **RemoveBuffer (** *MF_BUFFER_HANDLE* Buffer **)**	MultiFIFO buffer handle	None	Removes the MultiFIFO buffer
LPVOID **GetUserBuffer (** *MF_BUFFER_HANDLE* Buffer **)**	MultiFIFO buffer handle	Pointer to memory	Retrieves the pointer to the user data buffer from the MultiFIFO buffer handle
bool **ReadToBuffer (** *MF_BUFFER_HANDLE* Buffer, *DWORD *ReadSize,* *int Port* **)**	MultiFIFO buffer handle, Pointer to variable that will hold the size of data that was read, Port number	**true** if data transfer was successful; **false** if error occurred	Reads all the available data in the specified FIFO port into the **Buffer** (flushes the port)
bool **FillBuffer (** *MF_BUFFER_HANDLE* Buffer, *int Port* **)**	MultiFIFO buffer handle, Port number	**true** if data transfer was successful; **false** if error occurred	Reads the specified amount of data from the specified FIFO port into the **Buffer**
bool **Write (** *LPVOID Buffer,* *int Port,* *DWORD BufferSize = 0* **)**	Pointer to data buffer , Port number, Data length (optional, 0 if not used)	**true** if data transfer was successful; **false** if error occurred	Writes the data from the **Buffer** to the specified FIFO port

Table 2 – CMultiFIFO Effective Mode Methods

General Methods			
Method Name	**Input**	**Output**	**Description**
bool **IsDone (** *int* Port **)**	Port number	**true** if data transfer to/from the port is finished; **false** if the port is still busy	Checks whether the data transfer is finished for the specified port
bool **IsDone (** *MF_BUFFER_HANDLE* Buffer**)**	MultiFifo buffer handle	**true** if data transfer to/from the buffer is finished; **false** otherwise	Checks whether the data transfer is finished for the specified buffer
void **WaitDone (** *int* Port, *DWORD* Timeout **)**	Port number, Timeout in milliseconds	None	Waits until the data transfer is finished for the specified port or until the transfer times out
void **WaitDone (** *MF_BUFFER_HANDLE* Buffer, *DWORD* Timeout **)**	MultiFIFO buffer handle, Timeout in milliseconds	None	Waits until the data transfer is finished for the specified buffer or until the transfer times out
void **SetDMAChannel (** *DMA_CHANNEL* Channel **)**	DMA channel	None	Sets DMA channel for MultiFIFO operations
DWORD **GetDataSize (** *int* Port **)**	Port number	Free space left (for write FIFO); Amount of available data (for read FIFO).	Returns the amount of the available free space for write FIFO or the amount of the available data for read FIFO
MF_STATUS **GetLastError (** *void* **)**	None	Last error code that occurred during MultiFIFO operations	Use this method when one of the **CMultiFIFO** methods exits with an error
void **GetLastError (** *void* **)**	None	None	Resets the MultiFIFO and clears all its buffers

Table 3 – CMultiFIFO General Methods

To learn more about the Proc class methods, please refer to ***PROC API*** User's Manual.

The tables below describe the parameters and signals used in the Memory Controller subdesigns. These subdesigns are generated whenever a memory controller is used and are named using the following scheme: **IC_*X*_Bank_*Y*_Ctrl**, where ***X*** is the IC number and ***Y*** is the memory bank name. These HDL files (.tdf) are automatically generated by PROCWizard whenever the SDRAM controller is used. The signals that are driven from the user logic are automatically added to the user subdesigns, with the same name.

Notes

1. All GiDEL IPs should be connected using automatic generation with PROCWizard.
2. The parameters and the signals are defined automatically and there is no need to change them manually. The best way to change them is to define a new design in PROCWizard and generate a new HDL code.
3. There are several signals and parameters which are not described here. They are generated automatically to suit current design. The user should not change them in any case.

Parameters for SimpleFIFO

Parameter Name	Purpose	Comments
ADDR_WIDTH	Memory address bus width (in bits)	For example, if we define a 16MB FIFO and the memory cell is 4 bytes wide, then we have 4M address space and 22-bit address bus: ADDR_WIDTH = $Log_2(16M / 4) = 22$
RD_PORT_TYPE	Defines whether the port is connected to PCI or not	Possible values: PCI NON_PCI
WR_PORT_TYPE		

Table 4 - SimpleFIFO Parameter Names

Signal/Bus Names for SimpleFIFO

Signal / Bus Name *	Dir	Purpose	Comments
clrn	INPUT	Global reset signal	Global reset signal is active low.
mem_clk	INPUT	Memory clock	
lclk	INPUT	Local bus clock	
clk_**fifo_name**_src	INPUT	Source port clock	
clk_**fifo_name**_dst	INPUT	Destination port clock	
reset_**fifo_name**	INPUT	FIFO reset	Reset is an active high pulse. Should be at least 3 clocks wide (from corresponding reset source entity).
rd_req_**fifo_name**	INPUT	Request to read	
rd_ack_**fifo_name**	OUTPUT	Acknowledge for reading	
wr_req_**fifo_name**	INPUT	Request to write	
wr_ack_**fifo_name**	OUTPUT	Acknowledge for writing	
almost_full_**fifo_name**	OUTPUT	FIFO is almost full	Rises when there are less than 4K free cells in FIFO.
almost_empty_**fifo_name**	OUTPUT	FIFO is almost empty	Falls low when there are more than 4K free cells in FIFO.
half_full_**fifo_name**	OUTPUT	FIFO is half full flag	
fifo_name_flush	INPUT	End of transaction signal	Assert high when the last word of the transaction is transferred to FIFO.
data_**fifo_name**_dst[]	INPUT	Data from the read port	
data_**fifo_name**_src[]	OUTPUT	Data to the write port	

* **fifo_name** is the name given to the FIFO module during definition in PROCWizard.

Table 5 – SimpleFIFO Signal and Bus Names

Parameters for MultiFIFO

Parameter Name	Purpose	Comments
ADDR_WIDTH	Memory address bus width (in port words)	For example, if we define a 16MB FIFO and the memory cell is 4 bytes wide, then we have 4M address space and 22-bit address bus: ADDR_WIDTH = $Log_2(16M / 4) = 22$
FIFO_TYPE	Defines direction of the FIFO ports	Possible values: MULTI_RD => PCI port writes data MULTI_WR => PCI port reads data
NUMBER_OF_PORTS	Number of non-PCI ports	

Table 6 – MultiFIFO Parameter Names

Signal/Bus Names for MultiFIFO

The control signals below are related to the non-PCI side of the **MultiFIFO**. The GiDEL API automatically drives all other control/status signals.

Signal / Bus Name * / * *	Dir	Purpose	Comments
Clrn	INPUT	Global reset signal	Global reset signal is active low.
mem_clk	INPUT	Memory clock	
Lclk	INPUT	Local bus clock	
clk_**fifo_name**_**port_name**	INPUT	Non-PCI port clock	Each port may work in its own clock domain
req_**fifo_name**_**port_name**	INPUT	Request	Read / Write request depending on FIFO_TYPE
ack_**fifo_name**_**port_name**	OUTPUT	Acknowledge	Read / Write acknowledge depending on FIFO_TYPE

Signal / Bus Name * / * *	Dir	Purpose	Comments
almost_full_**fifo_name**_**port_name**	OUTPUT	FIFO is almost full flag	Each FIFO flag is related to its corresponding port_name.
almost_empty_**fifo_name**_**port_name**	OUTPUT	FIFO is almost empty flag	Each FIFO flag is related to its corresponding port_name
half_full_**fifo_name**_**port_name**	OUTPUT	FIFO is half full flag	Each FIFO flag is related to its corresponding port_name.

** **port_name** is the name given to the FIFO port during definition in PROCWizard.

** **fifo_name** is the name given to the FIFO module during definition in PROCWizard.

Table 7 – MultiFIFO Signal and Bus Names